Focused search for arguments from propositional knowledge

Vasiliki EFSTATHIOU [a] and Anthony HUNTER [a]

[a] *Department of Computer Science,*
University College London,
Gower Street, London WC1E 6BT, UK
{v.efstathiou, a.hunter}@cs.ucl.ac.uk

Abstract Classical propositional logic is an appealing option for modelling argumentation but the computational viability of generating an argument is an issue. Here we propose ameliorating this problem by harnessing the notion of a connection graph to reduce the search space when seeking all the arguments for a claim from a knowledgebase. For a set of clauses, a connection graph is a graph where each node is a clause and each arc denotes that there exist complementary disjuncts in the pair of nodes. For a set of formulae in conjunctive normal form, we use the notion of the connection graph for the set of clauses obtained from the conjuncts in the formulae. When seeking arguments for a claim, we can focus our search on a particular subgraph of the connection graph that we call the focal graph. Locating this subgraph is relatively inexpensive in terms of computational cost. In addition, using (as the search space) the formulae of the initial knowledgebase, whose conjuncts relate to this subgraph, can substantially reduce the cost of looking for arguments. We provide a theoretical framework and algorithms for this proposal, together with some theoretical results and some preliminary experimental results to indicate the potential of the approach.

1. Introduction

Argumentation is a vital aspect of intelligent behaviour by humans. Consider diverse professionals such as politicians, journalists, clinicians, scientists, and administrators, who all need to collate and analyse information looking for pros and cons for consequences of importance when attempting to understand problems and make decisions.

There are a number of proposals for logic-based formalisations of argumentation (for reviews see [7,18,4]). These proposals allow for the representation of arguments for and against some claim, and for counterargument relationships between arguments. In a number of key examples of argumentation systems, an argument is a pair where the first item in the pair is a minimal consistent set of formulae that proves the second item which is a formula (see for example [2,11,3,1,12]). Proof procedures and algorithms have been developed for finding preferred arguments from a knowledgebase using defeasible logic and following for example Dung's preferred semantics (see for example [5,20,17,14,6, 8,9]). However, these techniques and analyses do not offer any ways of ameliorating the computational complexity inherent in finding arguments for classical logic.

Suppose we use an automated theorem prover (an ATP). If we seek arguments for a particular claim α, we need to post queries to the ATP to ensure that a particular set of premises entails α, that the set of premises is minimal for this, and that it is consistent. So finding arguments for a claim α involves considering subsets Φ of Δ and testing them with the ATP to ascertain whether $\Phi \vdash \alpha$ and $\Phi \nvdash \bot$ hold. For $\Phi \subseteq \Delta$, and a formula α, let $\Phi?\alpha$ denote a call (a query) to an ATP. If Φ classically entails α, then we get the answer $\Phi \vdash \alpha$, otherwise we get the answer $\Phi \nvdash \alpha$. In this way, we do not give the whole of Δ to the ATP. Rather we call it with particular subsets of Δ. So for example, if we want to know if $\langle \Phi, \alpha \rangle$ is an argument, then we have a series of calls $\Phi?\alpha$, $\Phi?\bot$, $\Phi \setminus \{\phi_1\}?\alpha$,...,$\Phi \setminus \{\phi_k\}?\alpha$, where $\Phi = \{\phi_1, .., \phi_k\}$. So the first call is to ensure that $\Phi \vdash \alpha$, the second call is to ensure that $\Phi \nvdash \bot$, the remaining calls are to ensure that there is no subset Φ' of Φ such that $\Phi' \vdash \alpha$. This then raises the question of which subsets Φ of Δ to investigate when we are searching for an argument for α. Moreover, if we want to find all arguments for a claim in Δ, in the worst case we need to consider all subsets of Δ.

It is with these issues in mind that we explore an alternative way of finding all the arguments from a knowledgebase Δ for a claim α. Our approach is to adapt the idea of connection graphs to enable us to find arguments. A connection graph [15,16] is a graph where a clause is represented by a node and an arc (ϕ, ψ) denotes that there is a disjunct in ϕ with its complement being a disjunct in ψ. In previous work [10], we have proposed a framework for using connection graps for finding arguments. However, in that paper we restricted consideration to knowledgebases of clauses and claims to being literals. Furthermore, in that paper we did not give the algorithms for constructing the connection graphs, but rather we focused on algorithms for searching through graph structures for supports for arguments. Finally, we did not provide a systematic empirical evaluation of the algorithms for constructing graphs. We address these three shortcomings in this paper.

So, in this paper we propose algorithms for isolating a particular subset of the knowledgebase which essentially contains all the subsets of the knowledgebase that can be supports for arguments for a given claim. Initially we restrict the language used to a language of (disjunctive) clauses and we describe how arranging a clause knowledgebase in a connection graph structure can help focusing on a subgraph of the initial one that corresponds to the subset of a knowledgebase connected to a given clause. Furthermore, we illustrate how this approach can be generalised for a language of propositional formulae in conjunctive normal form where the clauses of interest in this case are the conjuncts of the negated claim for an argument. We describe how this method can be efficient regarding the search space reduction and hence, the computational cost of finding arguments. Finally, we present some experimental results illustrating how the software implementation of these algorithms performs.

2. Preliminaries

In this section, we review an existing proposal for logic-based argumentation [3] together with a recent proposal for using connection graphs in argumentation [10].

We consider a classical propositional language with classical deduction denoted by the symbol $\vdash$. We use $\Delta, \Phi, \ldots$ to denote sets of formulae, $\phi, \psi \ldots$ to denote formulae and $a, b, c\ldots$ to denote the propositional letters each formula consists of. For the following definitions we first assume a knowledgebase Δ (a finite set of formulae) and we use this Δ throughout. Furthermore we assume that each of the formulae of Δ is in conjunctive normal form (i.e. a conjunction of one or more disjunctive clauses). We use Δ as a large repository of information from which arguments can be constructed for and against arbitrary claims. The framework adopts a common intuitive notion of an argument. Essentially an argument is a set of relevant formulae from Δ that can be used to minimally and consistently entail a claim together with that claim. In this paper each claim is represented by a formula represented in conjunctive normal form.

Definition 1. *An* **argument** *is a pair* $\langle \Phi, \phi \rangle$ *such that* $(1)\ \Phi \subseteq \Delta, (2)\ \Phi \nvdash \bot, (3)\ \Phi \vdash \phi$ *and* (4) *there is no* $\Phi' \subset \Phi$ *s.t.* $\Phi' \vdash \phi$*.*

Example 1. *Let* $\Delta = \{\neg a, (\neg a \vee b) \wedge c, (d \vee e) \wedge f, \neg b \wedge d, (\neg f \vee g) \wedge (a \vee \neg e), \neg e \vee e, \neg k \vee m, \neg m\}$*. Some arguments are :* $\langle \{\neg a, (d \vee e) \wedge f\}, \neg a \wedge (d \vee e)\rangle, \langle \{(\neg a \vee b) \wedge c, \neg b \wedge d\}, \neg a \wedge c\rangle, \langle \{\neg a\}, \neg a\rangle, \langle \{\neg b \wedge d\}, d\rangle$*.*

We now turn to how the notion of connection graphs can be harnessed for focusing the search for arguments. In this section we restrict consideration to clause knowledgebases as follows.

Definition 2. *A language of clauses* $\mathcal{C}$ *is composed from a set of atoms* $\mathcal{A}$ *as follows: If* α *is an atom, then* α *is a* **positive literal***, and* $\neg\alpha$ *is a* **negative literal***. If* β *is a positive literal, or* β *is a negative literal, then* β *is a* **literal***. If* $\beta_1, .., \beta_n$ *are literals, then* $\beta_1 \vee ... \vee \beta_n$ *is a* **clause***. A* **clause knowledgebase** *is a set of clauses.*

We introduce relations on the elements of $\mathcal{C}$, that will be used to determine the links of graphs. We start by introducing the Disjuncts function which will be used for defining the attack relations between pairs of clauses.

Definition 3. *The* Disjuncts *function takes a clause and returns the set of disjuncts in the clause, and hence* $\mathsf{Disjuncts}(\beta_1 \vee .. \vee \beta_n) = \{\beta_1, .., \beta_n\}$*.*

Definition 4. *Let* ϕ *and* ψ *be clauses. Then,* $\mathsf{Preattacks}(\phi, \psi) = \{\beta \mid \beta \in \mathsf{Disjuncts}(\phi)$ *and* $\neg\beta \in \mathsf{Disjuncts}(\psi)\}$*.*

Example 2. $\mathsf{Preattacks}(a \vee \neg b \vee \neg c \vee d, a \vee b \vee \neg d \vee e) = \{\neg b, d\}$, $\mathsf{Preattacks}(a \vee b \vee \neg d \vee e, a \vee \neg b \vee \neg c \vee d) = \{b, \neg d\}$, $\mathsf{Preattacks}(a \vee b \vee \neg d, a \vee b \vee c) = \emptyset$, $\mathsf{Preattacks}(a \vee b \vee \neg d, a \vee b \vee d) = \{\neg d\}$, $\mathsf{Preattacks}(a \vee b \vee \neg d, e \vee c \vee d) = \{\neg d\}$.

Definition 5. *Let* ϕ *and* ψ *be clauses. If* $\mathsf{Preattacks}(\phi, \psi) = \{\beta\}$ *for some* β*, then* $\mathsf{Attacks}(\phi, \psi) = \beta$ *otherwise* $\mathsf{Attacks}(\phi, \psi) = null$*.*

Example 3. $\mathsf{Attacks}(a \vee \neg b \vee \neg c \vee d, a \vee b \vee \neg d \vee e) = null$, $\mathsf{Attacks}(a \vee b \vee \neg d, a \vee b \vee c) = null$, $\mathsf{Attacks}(a \vee b \vee \neg d, a \vee b \vee d) = \neg d$, $\mathsf{Attacks}(a \vee b \vee \neg d, e \vee c \vee d) = \neg d$.

Hence, the Preattacks relation is defined for any pair of clauses ϕ, ψ while the Attacks relation is defined for a pair of clauses ϕ, ψ for which $|\mathsf{Preattacks}(\phi, \psi)| = 1$.

We now introduce some types of graphs where each node corresponds to a clause and the links between each pair of clauses are determined according to the binary relations defined above. In the following examples of graphs we use the $|$, $\diagup$, $\diagdown$ and $—$ symbols to denote arcs in the pictorial representation of a graph.

Definition 6. *Let Δ be a clause knowledgebase. The* **connection graph** *for Δ, denoted* $\mathsf{Connect}(\Delta)$*, is a graph (N, A) where $N = \Delta$ and $A = \{(\phi, \psi) \mid$ there is a $\beta \in$* $\mathsf{Disjuncts}(\phi)$ *such that* $\beta \in \mathsf{Preattacks}(\phi, \psi)\}$.

Example 4. *The following is the connection graph for $\Delta = \{\neg b, \neg c \vee \neg g, \neg c, f \vee p, \neg l \vee \neg k, a \vee b, \neg b \vee d, c \vee g, \neg h \vee l, l \vee k, \neg a \vee d, \neg d, \neg g, h \vee \neg l, \neg k, n \vee m \vee \neg q, \neg m, \neg n, m, q\}$*

$$\begin{tikzcd}
\neg b \arrow[d, dash] & \neg c \vee \neg g \arrow[dr, dash] & \neg c \arrow[d, dash] & \neg h \vee l \arrow[r, dash] \arrow[d, dash] & \neg l \vee \neg k \arrow[d, dash] & & n \vee m \vee \neg q \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
a \vee b \arrow[r, dash] \arrow[d, dash] & \neg b \vee d \arrow[d, dash] & c \vee g \arrow[d, dash] & h \vee \neg l \arrow[r, dash] & l \vee k \arrow[d, dash] & \neg n & \neg m \arrow[d, dash] & q \\
\neg a \vee d \arrow[r, dash] & \neg d & \neg g & f \vee p & \neg k & & m &
\end{tikzcd}$$

The attack graph defined below is a subgraph of the connection graph identified using the Attacks function.

Definition 7. *Let Δ be a clause knowledgebase. The* **attack graph** *for Δ, denoted* $\mathsf{AttackGraph}(\Delta)$*, is a graph (N, A) where $N = \Delta$ and $A = \{(\phi, \psi) \mid$ there is a β* $\in \mathsf{Disjuncts}(\phi)$ *such that* $\mathsf{Attacks}(\phi, \psi) = \beta\}$.

Example 5. *Continuing Example 4, the following is the attack graph for Δ.*

$$\begin{tikzcd}
\neg b \arrow[d, dash] & \neg c \vee \neg g & \neg c \arrow[d, dash] & \neg h \vee l \arrow[r, dash] & \neg l \vee \neg k & & n \vee m \vee \neg q \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
a \vee b \arrow[r, dash] \arrow[d, dash] & \neg b \vee d \arrow[d, dash] & c \vee g \arrow[d, dash] & h \vee \neg l \arrow[r, dash] & l \vee k \arrow[d, dash] & \neg n & \neg m \arrow[d, dash] & q \\
\neg a \vee d \arrow[r, dash] & \neg d & \neg g & f \vee p & \neg k & & m &
\end{tikzcd}$$

The following definition of closed graph introduces a kind of connected subgraph of the attack graph where for each clause ϕ in the subgraph and for each disjunct β in ϕ there is another clause ψ in the subgraph such that $\mathsf{Attacks}(\psi, \phi) = \beta$ holds.

Definition 8. *Let Δ be a clause knowledgebase. The* **closed graph** *for Δ, denoted* $\mathsf{Closed}(\Delta)$*, is the largest subgraph (N, A) of* $\mathsf{AttackGraph}(\Delta)$*, such that for each $\phi \in N$, for each $\beta \in$* $\mathsf{Disjuncts}(\phi)$ *there is a $\psi \in N$ with* $\mathsf{Attacks}(\phi, \psi) = \beta$.

Example 6. *Continuing Example 5, the following is the closed graph for Δ.*

$$\begin{tikzcd}
\neg b \arrow[d, dash] & & \neg c \arrow[d, dash] & & n \vee m \vee \neg q \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
a \vee b \arrow[r, dash] \arrow[d, dash] & \neg b \vee d \arrow[d, dash] & c \vee g \arrow[d, dash] & \neg n & \neg m \arrow[d, dash] & q \\
\neg a \vee d \arrow[r, dash] & \neg d & \neg g & & m &
\end{tikzcd}$$

The focal graph (defined next) is a subgraph of the closed graph for Δ which is specified by a clause ϕ from Δ and corresponds to the part of the closed graph that contains ϕ. In the following, we assume a component of a graph means that each node in the component is connected to any other node in the component by a path.

Definition 9. *Let Δ be a clause knowledgebase and ϕ be a clause in Δ which we call the* **epicentre**. *The* **focal graph** *of ϕ in Δ denoted* $\mathsf{Focal}(\Delta, \phi)$ *is defined as follows: If* *there is a component X in* $\mathsf{Closed}(\Delta)$ *containing the node ϕ, then* $\mathsf{Focal}(\Delta, \phi) = X$*, otherwise* $\mathsf{Focal}(\Delta, \phi)$ *is the empty graph.*

Example 7. *Continuing Example 6, the following is the focal graph of $\neg b$ in Δ,*

$$\begin{tikzcd}
\neg b \arrow[d, dash] & \\
a \vee b \arrow[r, dash] \arrow[d, dash] & \neg b \vee d \arrow[d, dash] \\
\neg a \vee d \arrow[r, dash] & \neg d
\end{tikzcd}$$

The last example illustrates how the notion of the focal graph of an epicentre ϕ in Δ can be used in order to focus on the part of the knowledgebase that is relevant to ϕ. Later we will describe why the focal graph is important when it relates to a claim and how it can be used to reduce the search space when looking for arguments from propositional knowledge in conjunctive normal form.

Proposition 1. *Let Δ be a set of clauses. Then, $\forall \gamma_i, \gamma_j \in \Delta$, either* $\mathsf{Focal}(\Delta, \gamma_i)$ *and* $\mathsf{Focal}(\Delta, \gamma_j)$ *are the same component or they are disjoint components.*

3. Algorithm for finding the focal graph

In this section we present the algorithm (Algorithm 1) that returns the set of nodes of the focal graph of an epicentre ϕ in a clause knowledgebase Δ. The $\mathsf{GetFocal}(\Delta, \phi)$ algorithm finds the focal graph of ϕ in Δ by a depth first search following the links of the component of the $\mathsf{AttackGraph}(\Delta)$ that is linked to ϕ. For this, we have a data structure $Node_\psi$ (for each $\psi \in \Delta$) that represents the node for ψ in $\mathsf{AttackGraph}(\Delta)$. The attack graph can be represented by an adjacency matrix. Initially all the nodes are allowed as candidate nodes for the focal graph of ϕ in Δ, and then during the search they can be rejected if they do not satisfy the conditions of the definition for the focal graph. The algorithm chooses the appropriate nodes by using the boolean method $\mathsf{isConnected}(C, Node_\psi)$ which tests whether a node $Node_\psi$ of the attack graph $C = (N, A)$ is such that each literal $\beta \in \mathsf{Disjuncts}(\psi)$ corresponds to at least one arc to an allowed node (i.e. $\forall \beta \in \mathsf{Disjuncts}(\psi), \exists Node_{\psi'} \in \mathsf{AllowedNodes}$ s.t. $\mathsf{Attacks}(\psi, \psi') = \beta$), and so it returns false when there is a $\beta \in \mathsf{Disjuncts}(\psi)$ for which there is no $Node_{\psi'} \in \mathsf{AllowedNodes}$ s.t. $\mathsf{Attacks}(\psi, \psi') = \beta$. If this method does return false for a $Node_\psi$, then $Node_\psi$ is rejected and the algorithm backtracks to retest whether its adjacent allowed nodes are still connected. If some of them are no longer connected, they are rejected and in the same way their adjacent allowed nodes are tested recursively.

In the next section we show how this algorithm can be used for the generalised problem of finding arguments for any propositional formula when expressed in conjunctive normal form.

4. Using the focal graph algorithm for formulae in conjunctive normal form

To explain how the restricted language of clauses and Algorithm 1, can be used to deal with formulae (and thereby extend the proposal in [10]) we will first give some new

Algorithm 1 GetFocal(Δ, ϕ)

```
Let $C = (N, A)$ be the attack graph for $\Delta$ and $\phi$
Let AllowedNodes $= \{Node_\psi \mid \psi \in N\}$
Let VisitedNodes be the empty set.
if $\phi \notin \Delta$ or $\neg$isConnected$(C, Node_\phi)$ then
  return $\emptyset$
else
  Let S be an empty Stack
  push $Node_\phi$ onto S
end if
while S is not empty do
  Let $Node_\psi$ be the top of the stack S
  if $Node_\psi \in$ AllowedNodes then
    if isConnected$(C, Node_\psi)$ then
      if $Node_\psi \in$ VisitedNodes then
        pop $Node_\psi$ from S
      else
        VisitedNodes $=$ VisitedNodes $\cup \{Node_\psi\}$
        pop $Node_\psi$ from S
        for all $Node_{\psi'} \in$ AllowedNodes with Attacks$(\psi, \psi') \neq null$ do
          push $Node_{\psi'}$ onto S
        end for
      end if
    else
      AllowedNodes $=$ AllowedNodes $\setminus \{Node_\psi\}$
      VisitedNodes $=$ VisitedNodes $\cup \{Node_\psi\}$
      pop $Node_\psi$ from S.
      for all $Node_{\psi'} \in$ (AllowedNodes $\setminus$ VisitedNodes) with Attacks$(\psi, \psi') \neq null$
      do
        push $Node_{\psi'}$ onto S
      end for
    end if
  else
    pop $Node_\psi$ from S
  end if
end while
return AllowedNodes $\cap$ VisitedNodes
```

subsidiary definitions. For the following we assume a formula is in conjunctive normal form and a set of formulae contains formulae in conjunctive normal form.

Definition 10. *Let $\psi = \gamma_1 \wedge \ldots \wedge \gamma_n$ be a formula. The* Conjuncts(ψ) *function returns the clause knowledgebase $\{\gamma_1, \ldots, \gamma_n\}$.*

Example 8. *For $\phi = (a \vee b) \wedge (a \vee d \vee \neg c) \wedge \neg e$,* Conjuncts$(\phi) = \{a \vee b, a \vee d \vee \neg c, \neg e\}$.

Definition 11. *Let $\Phi = \{\phi_1, \ldots, \phi_k\}$ be a set of formulae. The* SetConjuncts(Φ) *function returns the union of all the conjuncts of the formulae from the set:* SetConjuncts$(\Phi) =$

$\bigcup_{\phi_i \in \Phi} \mathsf{Conjuncts}(\phi_i)$.

Example 9. *For* $\Phi = \{\neg a, (a \vee b) \wedge \neg d, (c \vee d) \wedge (e \vee f \vee \neg g), \neg d\}$, $\mathsf{SetConjuncts}(\Phi) = \{\neg a, a \vee b, \neg d, c \vee d, e \vee f \vee \neg g\}$.

Let $\psi = \delta_1 \wedge \ldots \wedge \delta_n$ be a formula and let $\overline{\psi}$ denote the conjunctive normal form of the negation of ψ, and so $\overline{\psi} = \gamma_1 \wedge \ldots \wedge \gamma_m \equiv \neg\psi$. Then, if we seek supports for arguments for ψ from a knowledgebase $\Phi = \{\phi_1, \ldots, \phi_k\}$, instead of searching among the arbitrary subsets of Φ, we can search among the subsets of Φ that consist of formulae whose conjuncts are contained in one of the focal graphs of each γ_i in $\mathsf{SetConjuncts}(\Phi \cup \{\overline{\psi}\})$. For this we need the notion of the $\mathsf{SubFocus}$ and the $\mathsf{SupportBase}$ defined next.

Definition 12. *Let* Φ *be a knowledgebase and* $\psi \in \Phi$*. Then for each* $\gamma_i \in \mathsf{Conjuncts}(\psi)$, $\mathsf{SubFocus}(\Phi, \gamma_i) = \mathsf{Focal}(\mathsf{SetConjuncts}(\Phi), \gamma_i)$.

Example 10. *Let* $\Phi = \{(a \vee b) \wedge (f \vee p) \wedge \neg c, (\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg d \wedge (\neg h \vee l), q \wedge (\neg h \vee l), c \vee g, \neg g, \neg b, \neg b \vee d, l \vee k, m \wedge (\neg l \vee \neg k), \neg k \wedge (n \vee m \vee \neg q), h \vee \neg l, \neg m \wedge \neg n, m \wedge q\}$*. Then,* $\mathsf{Conjuncts}(\Phi)$ *is equal to* Δ *from example 4. Let* $\phi = (a \vee b) \wedge (f \vee p) \wedge \neg c$*, and let* γ_1 *denote* $a \vee b$*,* γ_2 *denote* $f \vee p$ *and* γ_3 *denote* $\neg c$*. So, if* $\mathsf{SubFocus}(\Phi, \gamma_1) = (N_1, A_1)$, $\mathsf{SubFocus}(\Phi, \gamma_2) = (N_2, A_2)$ *and* $\mathsf{SubFocus}(\Phi, \gamma_3) = (N_3, A_3)$ *then* $N_1 = \{a \vee b, \neg a \vee d, \neg d, \neg b, \neg b \vee d\}$, $N_2 = \emptyset$*, and* $N_3 = \{\neg c, c \vee g, \neg g\}$.

The following definition introduces the notion of the query graph of a formula ψ in a knowledgebase Φ, which is a graph consisting of all the subfocuses of each of the $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$ in $\mathsf{Conjuncts}(\Phi \cup \{\overline{\psi}\})$. For a graph $C = (N, A)$ we let the function $\mathsf{Nodes}(C)$ return the set of clauses corresponding to the nodes of the graph (i.e. $\mathsf{Nodes}(C) = N$).

Definition 13. *Let* Φ *be a knowledgebase and* ψ *be a formula. The* **query graph** *of* ψ *in* Φ *denoted* $\mathsf{Query}(\Phi, \psi)$ *is the closed graph for the nodes*

$$\bigcup_{\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})} \mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i))$$

Example 11. *Let* $\Phi' = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg d \wedge (\neg h \vee l), q \wedge (\neg h \vee l), c \vee g, \neg g, \neg b, \neg b \vee d, l \vee k, m \wedge (\neg l \vee \neg k), \neg k \wedge (n \vee m \vee \neg q), (h \vee \neg l), \neg m \wedge \neg n, m \wedge q\}$ *and let* $\psi = (\neg a \vee \neg f \vee c) \wedge (\neg a \vee \neg p \vee c) \wedge (\neg b \vee \neg f \vee c) \wedge (\neg b \vee \neg p \vee c)$*. For* Φ' *and* ψ *we have* $\overline{\psi} = (a \vee b) \wedge (f \vee p) \wedge \neg c$*, which is equal to* ϕ *from Example 10 and* $\Phi' \cup \{\overline{\psi}\}$ *is equal to* Φ *from Example 10. Hence, continuing Example 10, the query graph of* ψ *in* Φ *is presented below and consists of the subgraphs* $(N_1, A_1), (N_2, A_2)$ *and* (N_3, A_3).

$$\begin{tikzcd}
\neg b \arrow[d, dash] & & \neg c \arrow[d, dash] \\
a \vee b \arrow[r, dash] \arrow[d, dash] & \neg b \vee d \arrow[d, dash] & c \vee g \arrow[d, dash] \\
\neg a \vee d \arrow[r, dash] & \neg d & \neg g
\end{tikzcd}$$

Hence, using each of the conjuncts γ_i of $\overline{\psi}$ as the epicentre for the focal graph in $\mathsf{SetConjuncts}(\Phi \cup \{\overline{\psi}\})$ we obtain the components of the query graph of ψ in Φ. The following definition introduces the notion of a zone, which relates each clause from each such subfocus to one or more formulae from knowledgebase Φ.

Definition 14. *Let Φ be a knowledgebase and ψ be a formula. Then, for each $\gamma_i \in$ $\mathsf{Conjuncts}(\overline{\psi})$,*

$$\mathsf{Zone}(\Phi, \gamma_i) = \{\phi \in \Phi \mid \mathsf{Conjuncts}(\phi) \cap \mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i) \neq \emptyset\}.$$

Example 12. *Continuing Example 11, for each $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi}), i = 1 \ldots 3$ we have* $\mathsf{Zone}(\Phi', \gamma_1) = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg b, \neg b \vee d, \neg d \wedge (\neg h \vee l)\}$, $\mathsf{Zone}(\Phi', \gamma_2) = \emptyset$ *and* $\mathsf{Zone}(\Phi', \gamma_3) = \{c \vee g, \neg g\}$.

The supportbase defined next, relates the clauses from the query graph of a formula ψ in a knowledgebase Φ to the corresponding set of formulae from Φ.

Definition 15. *For a knowledgebase Φ and a formula ψ the supportbase is given as follows:*

$$\mathsf{SupportBase}(\Phi, \psi) = \bigcup_{\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})} \mathsf{Zone}(\Phi, \gamma_i)$$

Example 13. *Continuing Example 11,* $\mathsf{SupportBase}(\Phi, \psi) = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg b, \neg b \vee d, \neg d \wedge (\neg h \vee l), c \vee g, \neg g\}$

According to the following proposition, the $\mathsf{SupportBase}(\Phi, \psi)$ defined above is the knowledgebase that contains all the arguments for ψ from Φ.

Proposition 2. *Let Φ be a knowledgebase and ψ be a formula. If $\langle \Psi, \psi \rangle$ is an argument from Φ and $\Psi \subseteq \Phi$, then $\Psi \subseteq \mathsf{SupportBase}(\Phi, \psi)$.*

Hence, by Proposition 2 it follows that instead of using the power set of the initial knowledgebase Φ in order to look for arguments for ψ, we can use the power set of $\mathsf{SupportBase}(\Phi, \psi)$. Using these definitions, we can introduce the additional algorithms that delineate the extension of the language of clauses used in previous sections and the use of Algorithm 1 to support a language of propositional formulae in conjunctive normal form. These are Algorithm 2 and Algorithm 3. Since our problem is focused on searching for arguments for a claim ψ, the part of the knowledgebase that we want to isolate will be delineated by ψ and, in particular, by the conjuncts of $\overline{\psi}$. Algorithm 2 returns the query graph of ψ in Φ as a set containing all the $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i)$ components for each $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$. Then, Algorithm 3 uses these results in order to retrieve the set of formulae from the initial knowledegbase to which each such set of conjuncts corresponds. So, Algorithm 3 returns a set of sets, each of which represents a zone for each $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$. The union of all these sets will be $\mathsf{SupportBase}(\Phi, \psi)$.

In Algorithm 2 we can see that it is not always necessary to use Algorithm 1 for each of the $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$ when trying to isolate the appropriate subsets of Φ. Testing for containment of a clause $\gamma_j \in \mathsf{Conjuncts}(\overline{\psi})$ in an already retrieved set $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i), i < j$ (where the ordering of the indices describes the order in which the algorithm is applied for each of the conjuncts) is enough to give the $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j)$ according to proposition 1. Furthermore, according to the following proposition, conjuncts of $\overline{\psi}$ within the same $\mathsf{SubFocus}$ correspond to the same set of formulae from Φ.

Algorithm 2 $\mathsf{GetQueryGraph}(\Phi, \psi)$

```
Let $\overline{\psi}$ be $\neg\psi$ in CNF : $\overline{\psi} \equiv \gamma_1 \wedge \ldots \wedge \gamma_m$
Let $S$ be a set to store sets of clauses, initially empty
Let $\mathsf{Clauses} = \mathsf{SetConjuncts}(\Phi \cup \{\overline{\psi}\})$
for $i = 1 \ldots m$ do
  if $\exists S_j \in S$ s.t. $\gamma_i \in S_j$ then
    $i = i + 1$
  else
    $S_i = \mathsf{GetFocal}(\mathsf{Clauses}, \gamma_i)$
  end if
  $S = S \cup \{S_i\}$
end for
return $S$
```

Proposition 3. *Let Φ be a set of formulae and let ψ be a formula. If* $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i) = \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j)$ *for some* $\gamma_i, \gamma_j \in \mathsf{Conjuncts}(\overline{\psi})$, *then* $\mathsf{Zone}(\Phi, \gamma_i) = \mathsf{Zone}(\Phi, \gamma_j)$.

The converse of the last proposition does not hold as the following example illustrates.

Example 14. *For* $\Phi = \{(c \vee g) \wedge d, d \vee f, \neg q, (d \vee p) \wedge f, \neg n, k \vee \neg m\}$ *and* $\psi = c \vee g \vee d$ *we have* $\overline{\psi} = \neg c \wedge \neg g \wedge \neg d$ *and* $N_1 \equiv \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \neg c) = \{\neg c, \neg g, c \vee g\}$, $N_2 \equiv \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \neg g) = \{\neg c, \neg g, c \vee g\} = N_1$ *and* $N_3 \equiv \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \neg d) = \{\neg d, d\}$. *Furthermore,* $\mathsf{Zone}(\Phi, \neg c) = \mathsf{Zone}(\Phi, \neg g) = \mathsf{Zone}(\Phi, \neg d) = \{(c \vee g) \wedge d\}$ *although* $N_3 \neq N_2$ *and* $N_3 \neq N_1$.

Hence, conjuncts of $\overline{\psi}$ with the same focal graph correspond to the same supportbase. Taking this into account, the following algorithm retrieves all the supportbases for each $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$.

Algorithm 3 $\mathsf{RetrieveZones}(\Phi, \psi)$

```
Let $Z$ be a set to store sets of formulae, initially empty
Let $S = \mathsf{GetQueryGraph}(\Phi, \psi) \equiv \{S_1, \ldots, S_k\}$
for $i = 1 \ldots k$ do
  Let $Z_i$ be the emptyset.
  for $j = 1 \ldots |S_i|$ do
    Let $\gamma_j$ be the j-th element of $S_i$
    Let $C_j = \{\phi \in \Phi \mid \gamma_j \in \mathsf{Conjuncts}(\phi)\}$
    $Z_i = Z_i \cup C_j$
  end for
  $Z = Z \cup \{Z_i\}$
end for
return $Z$
```

So, Algorithm 3 returns a set of sets, corresponding to all the possible zones identified by all the $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$. The union of all these sets will be $\mathsf{SupportBase}(\Phi, \psi)$.

Instead of looking for arguments for ψ among the arbitrary subsets of $\mathsf{SupportBase}(\Phi, \psi)$, we can search the power set of each non empty $\mathsf{Zone}(\Phi, \gamma_i)$ separately as the following proposition indicates.

Proposition 4. *Let Φ be a knowledgebase and ψ be a clause. If $\langle \Psi, \psi \rangle$ is an argument from Φ, then there is a $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$ s.t $\Psi \subseteq \mathsf{Zone}(\Phi, \gamma_i)$.*

In future work we plan to address the efficiency of using the subsets of each zone separately instead of using the power set of the supportbase in our search for arguments. We conjecture that this will improve the performance on average for finding arguments.

5. Experimental results

This section covers a preliminary experimental evaluation of algorithm 1 using a prototype implementation programmed in java running on a modest PC (Core2 Duo 1.8GHz).

The experimental data were obtained using randomly generated clause knowledgebases of a fixed number of 600 clauses according to the fixed clause length model K-SAT ([19,13]) where the chosen length (i.e. K) for each clause was either a disjunction of 3 literals or a disjunction of 1 literal. The clauses of length 3 can be regarded as **rules** and clauses of length 1 as **facts**. Each disjunct of each clause was randomly chosen out of a set of N distinct variables (i.e. atoms) and negated with probability 0.5.

In the experiment, we considered two dimensions. The first dimension was the clauses-to-variables ratio. For the definition of this ratio we take the integer part of the division of the number of clauses in Δ by the number of variables N (i.e. $\lfloor |\Delta|/|N| \rfloor$). The second dimension was the proportion of facts-to-rules in the knowledgebases tested. The preliminary results are presented in Figure 1 where each curve in the graph corresponds to one of those variations on the proportion of facts-to-rules. More precisely, each curve relates to one of the following (n, n') tuples where n represents the number of facts and n' represents the number of rules in the set: (150,450), (200,400), (300,300), (400,200), (450,150). Since each clause knowledgebase contains 600 elements, each of these tuples sums to 600. Each point on each curve is the average focal graph size from 1000 repetitions of running Algorithm 1 for randomly generated epicentres and randomly generated knowledgebases of a fixed clauses-to-variables ratio represented by coordinate x. For the results presented, since the values on axis x ranges from 1 to 15, the smallest number of variables used throughout the experiment was 40 which corresponds to a clauses-to-variables ratio of 15, while the largest number of variables used was 600 which corresponds to a clauses to variables ratio of 1.

The evaluation of our experiment was based on the size of the focal graph of an epicentre in a clause knowledgebase compared to the cardinality of the knowledgebase. For a fixed number of clauses, the number of distinct variables that occur in the disjuncts of all these clauses determines the size of the focal graph. In Figure 1 we see that as the clauses-to-variables ratio increases, the average focal graph size also increases because an increasing clauses-to-variables ratio for a fixed number of clauses implies a decreasing number of variables and this allows for a distribution of the variables amongst the clauses such that it is more likely for a literal to occur in a clause with its opposite occuring in another clause. We have noticed in previous experiments [10] that for a clause knowledgebase consisting of 3-place clauses only, an increasing clauses-to-variables ra-

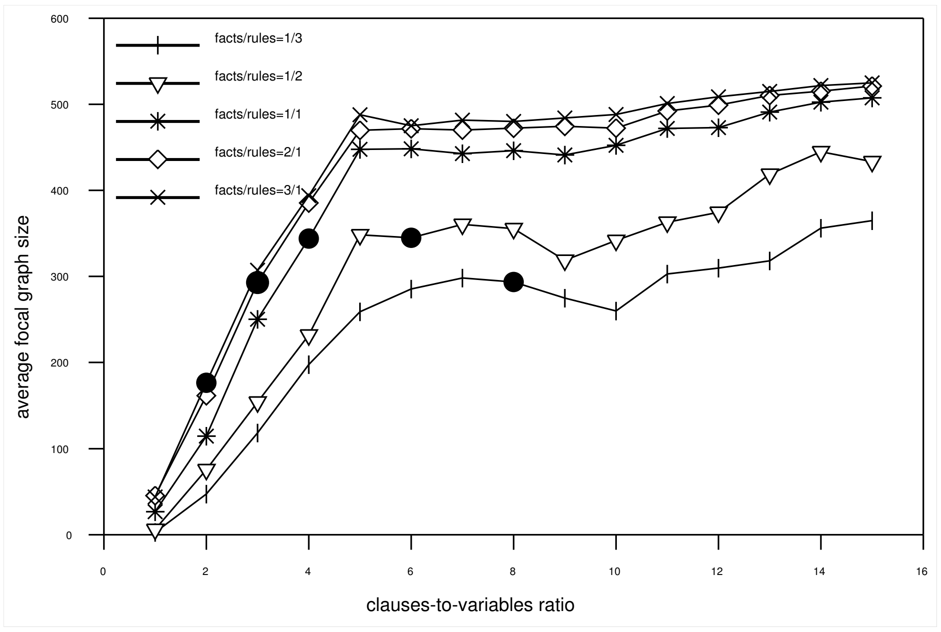

Figure 1. Focal graph size variation with the clauses-to-variables ratio

tio in the range $[5, 10]$ ameliorates the performance of the system as it increases the probability of a pair of clauses ϕ, ψ from Δ being such that $|\mathsf{Preattacks}(\phi, \psi)| > 1$. This is because a ratio in this range makes the occurrence of a variable and its negation in the clauses of the set so frequent that it allows the $\mathsf{Attacks}$ relation to be defined only on a small set of clauses from the randomly generated clause knowledgebase. In the graph presented in this paper though, this is less likely to happen as the clause knowledgebases tested involve both facts and rules.

Including literals (facts) in the knowledgebases used during the experiment makes the repeated occurrence of the same fact and its complement in the randomly generated clause knowledgebases, and hence in the subsequent focal graph, more frequent. It is for this reason that the curves with lower facts-to-rules proportion have a lower average focal graph (for each clauses-to-variables ratio). The symbol $\bullet$ on each of the curves indicates the highest possible clauses-to-variables ratio that would allow for a randomly generated clause knowledgebase consisting of the corresponding proportion of facts and rules to contain only distinct elements. Hence, for the data presented in this graph, the largest average focal graph of a randomly generated clause in a randomly generated clause knowledgebase of 600 distinct elements has the value 343.99 which corresponds to 57% of the initial knowledgebase. The values of the parameters with which this maximum is obtained correspond to a clauses-to-variables ratio equal to 4 on knowledgebases with a 1 to 1 proportion of facts-to-rules.

The average time for each repetition of the algorithm ranged from 6.3 seconds (for a facts-to-rules proportion of 1-3) to 13.8 seconds (for a facts-to-rules proportion of 3-1). So, the results show that for an inexpensive process we can substantially reduce the search space for arguments.

6. Discussion

In this paper, we have proposed the use of a connection graph approach as a way of ameliorating the computation cost when searching for arguments. We have extended the theory and algorithms proposed in previous work [10] for a language of clauses so as to deal with any set of propositional formulae provided that these are represented in conjunctive normal form. We have provided theoretical results to ensure the correctness of the proposal, and we have provided provisional empirical results to indicate the potential advantages of the approach.

References

[1] L. Amgoud and C. Cayrol. A model of reasoning based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34:197–216, 2002.

[2] S. Benferhat, D. Dubois, and H. Prade. Argumentative inference in uncertain and inconsistent knowledge bases. In *Proceedings of the 9th Annual Conference on Uncertainty in Artificial Intelligence (UAI 1993)*, pages 1449–1445. Morgan Kaufmann, 1993.

[3] Ph. Besnard and A. Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128:203–235, 2001.

[4] Ph. Besnard and A. Hunter. *Elements of Argumentation*. MIT Press, 2008.

[5] D. Bryant, P. Krause, and G. Vreeswijk. Argue tuProlog: A lightweight argumentation engine for agent applications. In *Computational Models of Argument (Comma'06)*, pages 27–32. IOS Press, 2006.

[6] C. Cayrol, S. Doutre, and J. Mengin. Dialectical proof theories for the credulous preferred semantics of argumentation frameworks. In *Quantitative and Qualitative Approaches to Reasoning with Uncertainty*, volume 2143 of *LNCS*, pages 668–679. Springer, 2001.

[7] C. Chesñevar, A. Maguitman, and R. Loui. Logical models of argument. *ACM Computing Surveys*, 32:337–383, 2000.

[8] Y. Dimopoulos, B. Nebel, and F. Toni. On the computational complexity of assumption-based argumentation for default reasoning. *Artificial Intelligence*, 141:57–78, 2002.

[9] P. Dung, R. Kowalski, and F. Toni. Dialectical proof procedures for assumption-based admissible argumentation. *Artificial Intelligence*, 170:114–159, 2006.

[10] V. Efstathiou and A. Hunter. Algorithms for effective argumentation in classical propositional logic. In *Proceedings of the International Symosium on Foundations of Information and Knowledge Systems (FOIKS 2008)*, LNCS. Springer, 2008.

[11] M. Elvang-Gøransson, P. Krause, and J. Fox. Dialectic reasoning with classically inconsistent information. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence (UAI 1993)*, pages 114–121. Morgan Kaufmann, 1993.

[12] A. García and G. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.

[13] I. P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70(1-2):335–345, 1994.

[14] A. Kakas and F. Toni. Computing argumentation in logic programming. *Journal of Logic and Computation*, 9:515–562, 1999.

[15] R. Kowalski. A proof procedure using connection graphs. *Journal of the ACM*, 22:572–595, 1975.

[16] R. Kowalski. *Logic for problem solving*. North-Holland Publishing, 1979.

[17] H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 7:25–75, 1997.

[18] H. Prakken and G. Vreeswijk. Logical systems for defeasible argumentation. In D. Gabbay, editor, *Handbook of Philosophical Logic*. Kluwer, 2000.

[19] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

[20] G. Vreeswijk. An algorithm to compute minimally grounded and admissible defence sets in argument systems. In *Computational Models of Argument (Comma'06)*, pages 109–120. IOS Press, 2006.